**Kubernetes**

**What Is Kubernetes**

**Kubernetes** is an open-source platform designed to automate deploying, scaling, and operating **application** containers. With **Kubernetes**, you are able to quickly and efficiently respond to customer demand: Deploy your applications quickly and predictably.

what is Kubernetes. Kubernetes is an open source platform that automates container operations such as deploying containers, scaling up, scaling down of containers, load balancing etc. You can just cluster the hosts running containers and Kubernetes will help you easily and efficiently manage these clusters

**What is a docker**

**Docker container** is an open source software development platform. Its main benefit is to package applications in “**containers**,” allowing them to be portable among any system running the Linux operating system (OS).

**What is orchestration in software?**

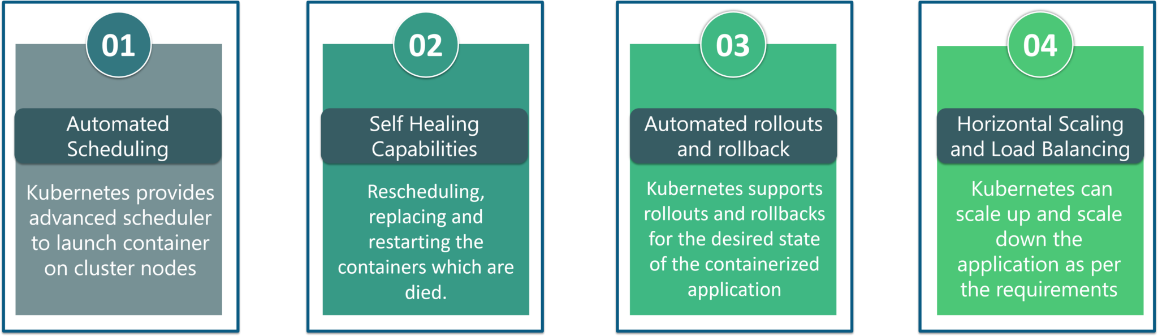
Application Orchestration. Application or service orchestration is the process of integrating two or more applications and/or services together to automate a process, or synchronize data in real-time. Often, point-to-point integration may be used as the path of least resistance.

**Why Kubernetes:**

* Each service’s monitoring is difficult.
* Scaling a particular service based on load is not possible.
* Too much manual intervention required for managing containers.
* Managing containers on multiple servers become difficult.

**What does Kubernetes do:**

The features of Kubernetes, are as follows:



* **Automated Scheduling:** Kubernetes provides advanced scheduler to launch container on cluster nodes based on their resource requirements and other constraints, while not sacrificing availability.
* **Self Healing Capabilities:** Kubernetes allows to replaces and reschedules containers when nodes die. It also kills containers that don’t respond to user-defined health check and doesn’t advertise them to clients until they are ready to serve.
* **Automated rollouts & rollback:** Kubernetes rolls out changes to the application or its configuration while monitoring application health to ensure it doesn’t kill all your instances at the same time. If something goes wrong, with Kubernetes you can rollback the change.
* **Horizontal Scaling & Load Balancing:** Kubernetes can scale up and scale down the application as per the requirements with a simple command, using a UI, or automatically based on CPU usage.

**Compare Docker and Kubernetes:**

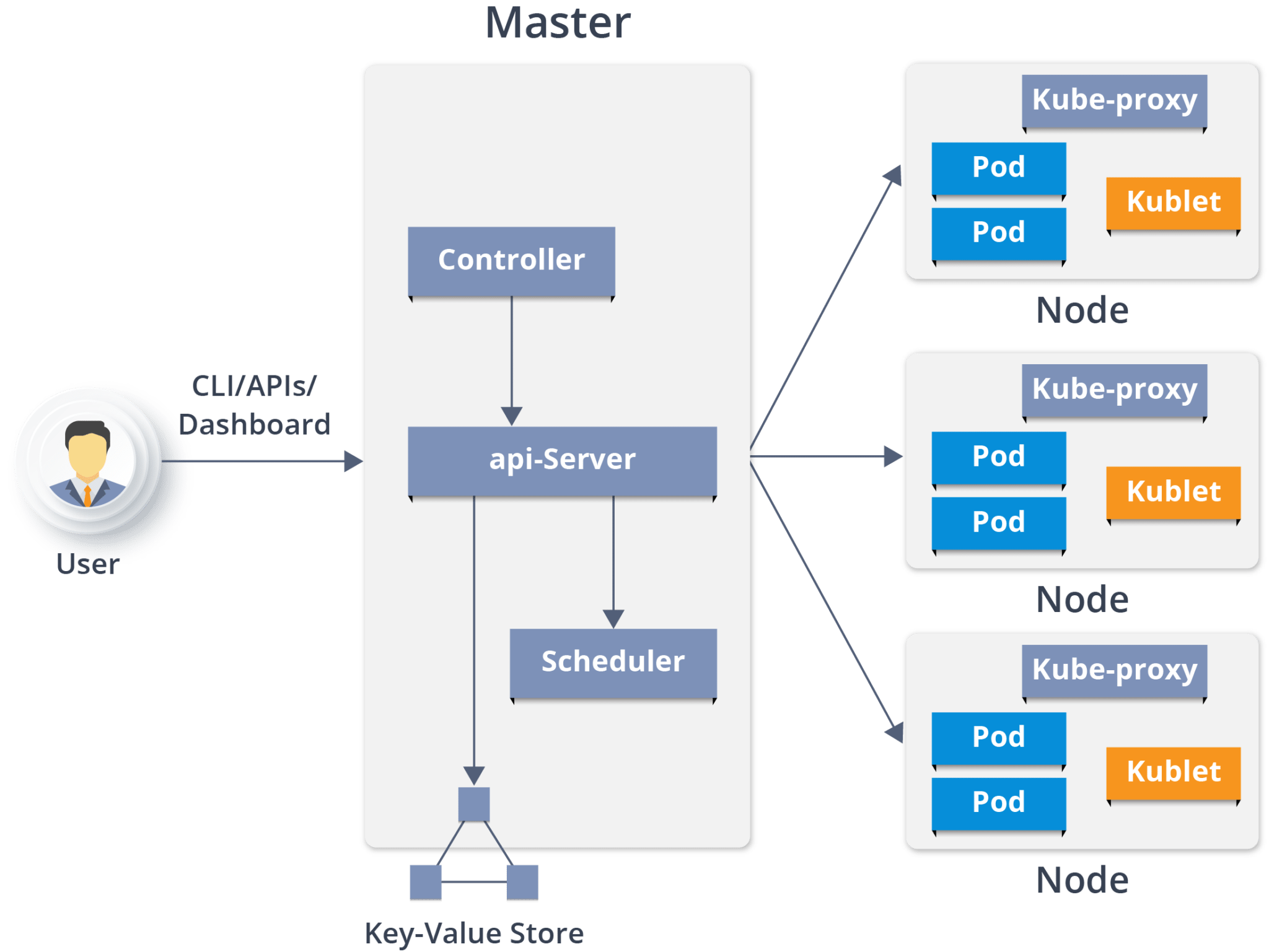
Docker: docker is a containerization platform.

### Kubernetes: is container management platform, which means that once we have containerized of application with help of docker containers or linux containers and when scaling of those containers to a big number like 50 to 100, this situation comes in kubernetes, so when we multiple containers need to managed kubernetes helps.

### How is Kubernetes different from Docker Swarm?

|  |  |  |
| --- | --- | --- |
| **Features** | **Kubernetes** | **Docker Swarm** |
| **Installation & Cluster Config** | Setup is very complicated, but once installed cluster is robust. | Installation is very simple, but the cluster is not robust. |
| **GUI** | GUI is the Kubernetes Dashboard. | There is no GUI. |
| **Scalability** | Scaling up is slow compared to swarm  but guarantee strong cluster sate | Scaling up faster than Kubernetes, but cluster  Strength not as robust. |
| **Auto-scaling** | Kubernetes can do auto-scaling. | Docker swarm cannot do auto-scaling. |
| **Load Balancing** | Loan balancing Manual confingurationrequired for load balancing traffic between different containers and pods. | Docker swarm does auto load balancing of traffic between containers in the cluster. |
| **Rolling Updates & Rollbacks** | Can deploy rolling updates and does automatic rollbacks. | Can deploy rolling updates, but not automatic rollback. |
| **DATA Volumes** | Can share storage volumes only with the other containers in the same pod.  Pod is nothing but group related containers  , logical group of containers together | Can share storage volumes with any other container. |
| **Logging & Monitoring** | In-built tools for logging and monitoring. | 3rd party tools like ELK stack should be used for logging and monitoring. |

## ****Kubernetes Architecture/Kubernetes Components****



Kubernetes Architecture has the following main components:

* Master nodes
* Worker/Slave nodes
* Distributed key-value store(etcd.)

## Master Node

It is the entry point for all administrative tasks which is responsible for managing the Kubernetes cluster. There can be more than one master node in the cluster to check for fault tolerance. More than one master node puts the system in a High Availability mode, in which one of them will be the main node which we perform all the tasks.

For managing the cluster state, it uses etcd in which all the master nodes connect to it.

Master is the central control point that provides a unified view of the cluster. There is a single master node that control multiple minions.

Master servers work together to accept user requests, determine the best ways to schedule workload containers, authenticate clients and nodes, adjust cluster-wide networking, and manage scaling and health checking responsibilities

**API server:**

API server is the frontend into the Kubernetes control plane.

* Performs all the administrative tasks through the API server within the master node.
* In this REST commands are sent to the API server which validates and processes the requests.
* After requesting, the resulting config and state of the cluster is stored in the distributed key-value store.
* Cluster store is its memory, the config and state of the cluster gets pertinently stored in the cluster store

Kubernetes API server is the central management entity that receives all REST requests for modifications (to pods, services, replication sets/controllers and others), serving as frontend to the cluster. Also, this is the only component that communicates with the etcd cluster, making sure data is stored in etcd and is in agreement with the service details of the deployed pods.

**Scheduler:**

* The scheduler schedules the tasks to slave nodes. It stores the resource usage information for each slave node.
* It schedules the work in the form of Pods and Services.
* Before scheduling the task, the scheduler also takes into account the quality of the service requirements, data locality, affinity, anti-affinity, etc.

**Controller manager:**

* Also known as controllers.
* It is a daemon which regulates the Kubernetes cluster which manages the different non-terminating control loops.
* It also performs lifecycle functions such as namespace creation and lifecycle, event garbage collection, terminated-pod garbage collection, cascading-deletion garbage collection, node garbage collection, etc.
* Basically, a controller watches the desired state of the objects it manages and watches their current state through the API server. If the current state of the objects it manages does not meet the desired state, then the control loop takes corrective steps to make sure that the current state is the same as the desired state.

**What is the ETCD?**

* etcd is a distributed key-value store which stores the cluster state.
* It can be part of the Kubernetes Master, or, it can be configured externally.
* etcd is written in the Go programming language. In Kubernetes, besides storing the cluster state (based on the Raft Consensus Algorithm) it is also used to store configuration details such as subnets, ConfigMaps, Secrets, etc.
* A raft is a consensus algorithm designed as an alternative to Paxos. The Consensus problem involves multiple servers agreeing on values; a common problem that arises in the context of replicated state machines. Raft defines three different roles (Leader, Follower, and Candidate) and achieves consensus via an elected leader

Now you have understood the functioning of Master node. Let’s see what is the Worker/Minions node and its components.

## Worker Node (formerly minions)

It is a physical server or you can say a VM which runs the applications using Pods (a pod scheduling unit) which is controlled by the master node. On a physical server (worker/slave node), pods are scheduled. For accessing the applications from the external world, we connect to nodes.

Let’s see what are the following components:

**Container runtime:**

* To run and manage a container’s lifecycle, we need a container runtime on the worker node.
* Sometimes, Docker is also referred to as a container runtime, but to be precise, Docker is a platform which uses containers as a container runtime.

**Kubelet:**

* It is an agent which communicates with the Master node and executes on nodes or the worker nodes. It gets the Pod specifications through the API server and executes the containers associated with the Pod and ensures that the containers described in those Pod are running and healthy.

**Kube-proxy:**

### Pods

* A pod is one or more containers that logically go together. Pods run on nodes. Pods run together as a logical unit. So they have the same shared content. They all share the same IP address but can reach other Pods via localhost, as well as shared storage. Pods don’t need to all run on the same machine as containers can span more than one machine. One node can run multiple pods.

**Conclusion**

* **API server**: The entry point for all REST commands, the sole component of the Master Node which is user-accessible.
* **Datastore:** Strong, consistent, and highly-available key-value storage used by the Kubernetes cluster.
* **Scheduler:** Watches for newly-created pods and assigns them to nodes. Deployment of pods and services onto the nodes happen because of the scheduler.
* **Controller manager:** Runs all the controllers that handle routine tasks in the cluster.
* **Worker nodes**: Primary node agent, also called minion nodes. The pods are run here. Worker nodes contain all the necessary services to manage networking between the containers, communicate with the master node, and assign resources to the containers scheduled.
* **Docker:** Runs on each worker node and downloads images and starting containers.
* **Kubelet**: Monitors the state of a pod and ensures that the containers are up and running. It also communicates with the data store, getting information about services and writing details about newly created ones.
* **Kube-proxy:** A network proxy and load balancer for a service on a single worker node. It is responsible for traffic routing.
* **Kubectl:** A CLI tool for the users to communicate with the Kubernetes API server.

**Deployement:**

A deployment is used to keep a set of pods running by creating pods from a template.

**Service:**

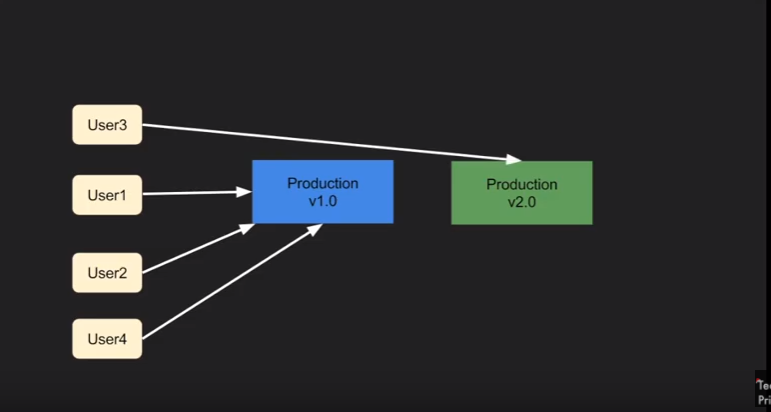
A service is used to allow network access to a set of pods.

**Blue-green deployments**

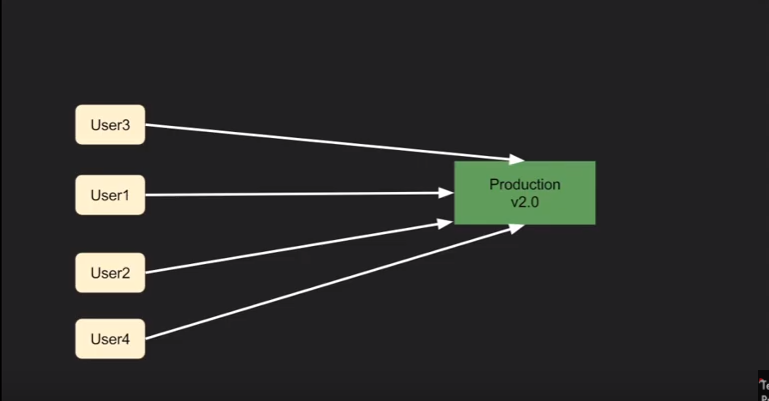
Blue-green deployments are a pattern whereby we reduce downtime during production deployments by having two production environments ("blue" and "green").

Blue is current production environment and green is new version will deployed.

We can forward some specific user request to green deployment,



Once the software is working in the green environment, you switch the router so that all incoming requests go to the green environment - the blue one is now idle.



**Step 1: cloning the app**

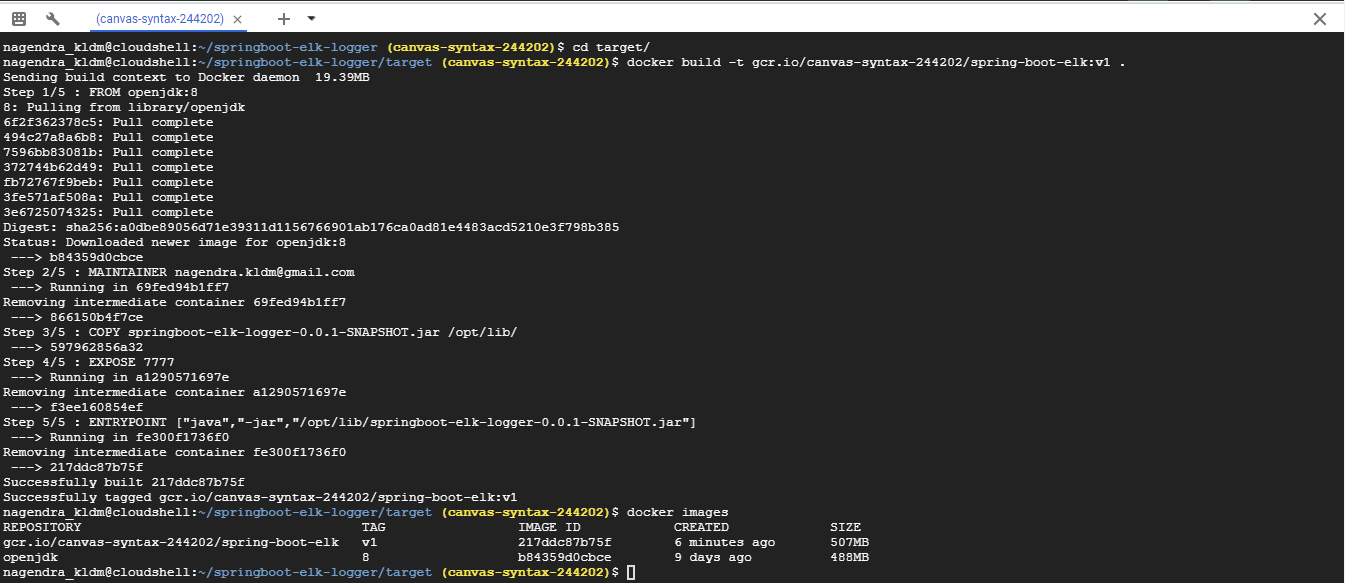
git clone "https://github.com/NagendraMekala/springboot-elk-logger.git"

**Step 2: make executable package**

mvn clean package

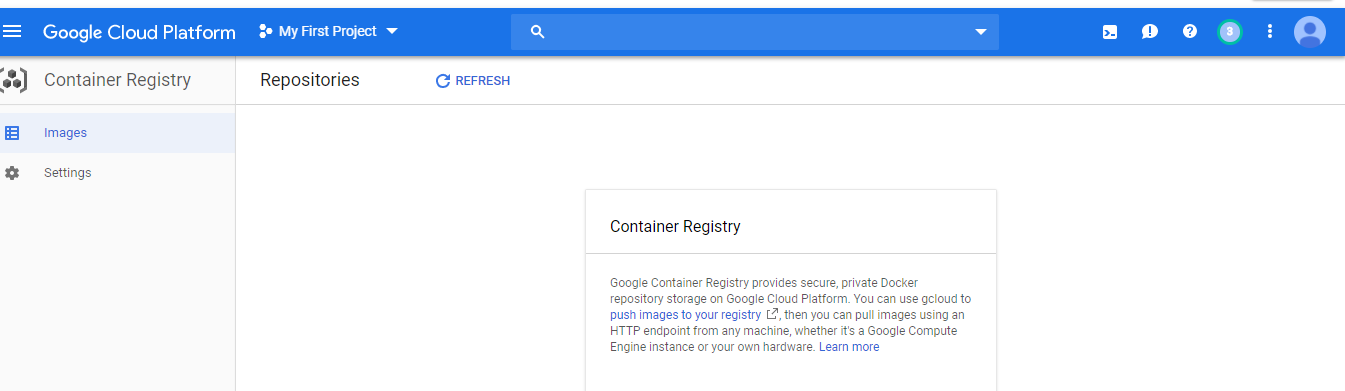
**Step 3: build the docker image**

docker build -t gcr.io/canvas-syntax-244202/spring-boot-elk:v1 .

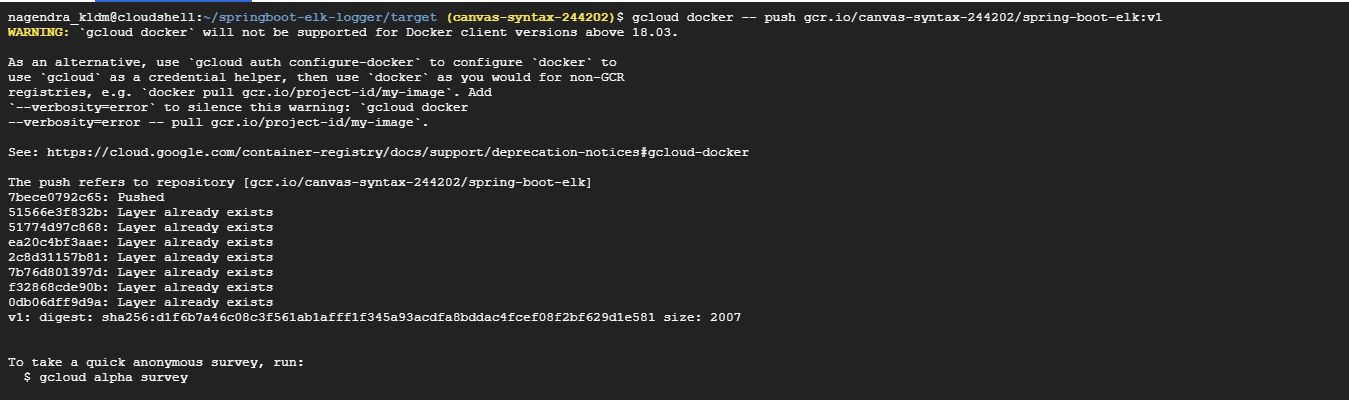


**Step 3: pushing the docker image into container registry**

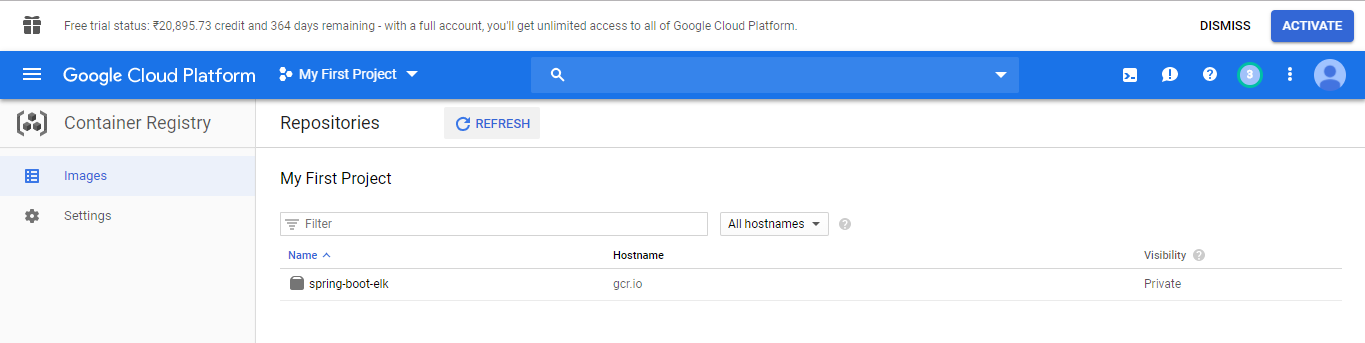
Before pushing



gcloud docker -- push gcr.io/canvas-syntax-244202/spring-boot-elk:v1

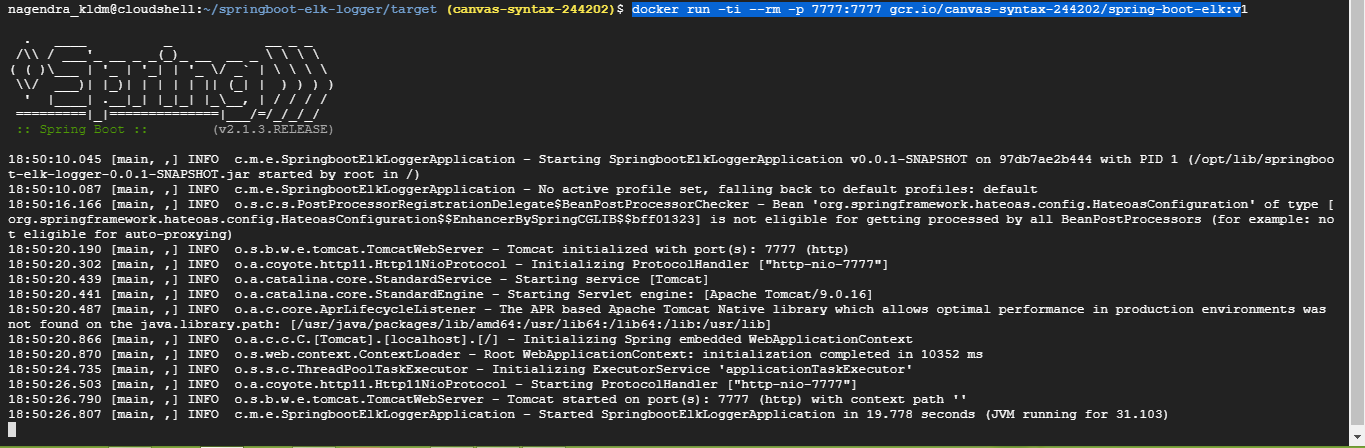


After pushing



**Step 4: verify the docker image working or not**

docker run -ti --rm -p 7777:7777 gcr.io/canvas-syntax-244202/spring-boot-elk:v1

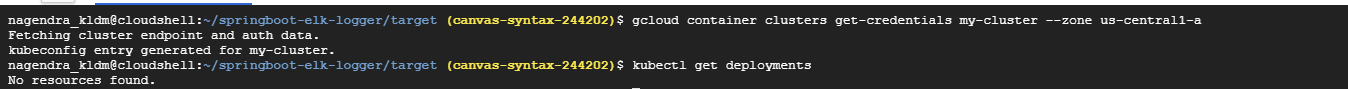




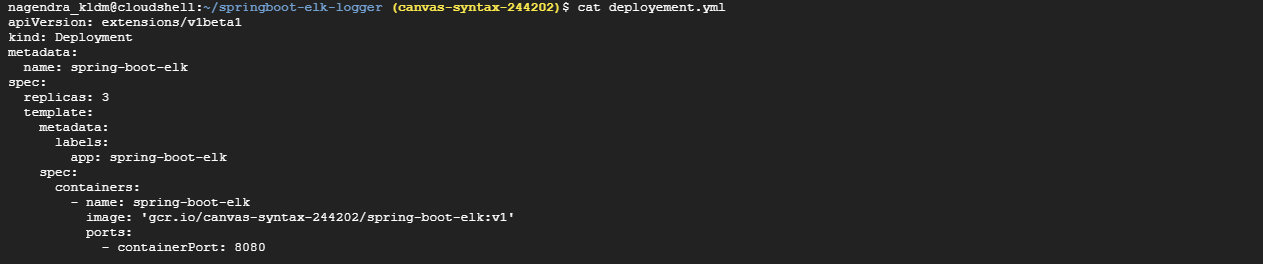
**Step 5: Login to the K8s Cluster**

Command to login to the K8s cluster from Cloud Shell

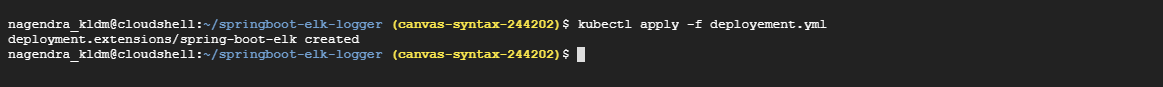
gcloud container clusters get-credentials my-cluster --zone us-central1-a



**Step 6: creating deployment yml file**



**Step 7: create deployment**

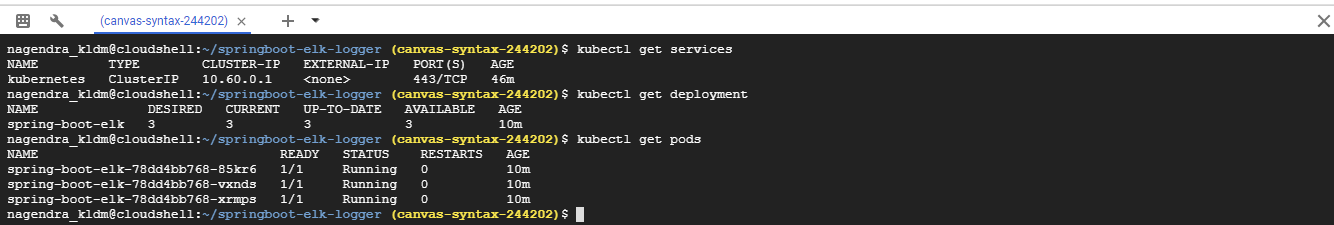
****

**Step 8: Verify the kubectl commands**

kubectl get deployments

kubectl get pods

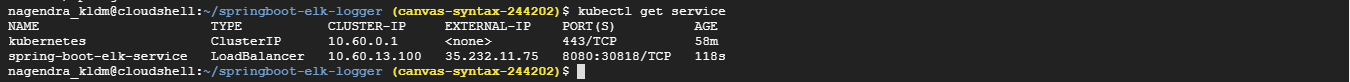
kubectl get services

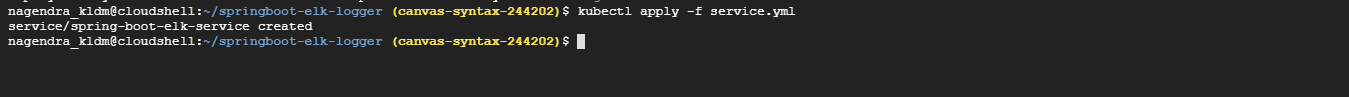
****

**Step 9: create service**

kubectl get service

kubectl apply -f service.yml

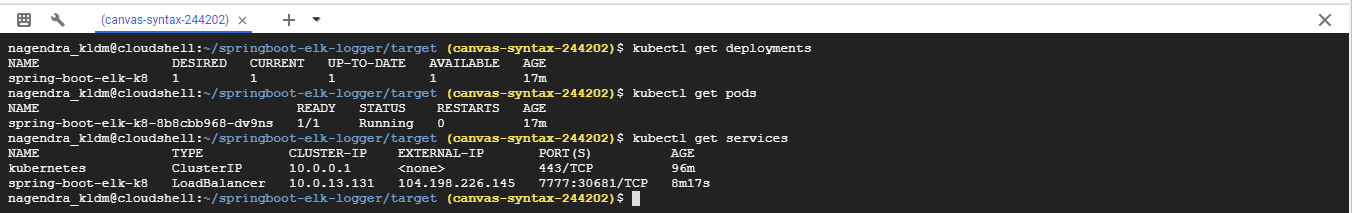
****

****

**Step 10: Scale deployments**

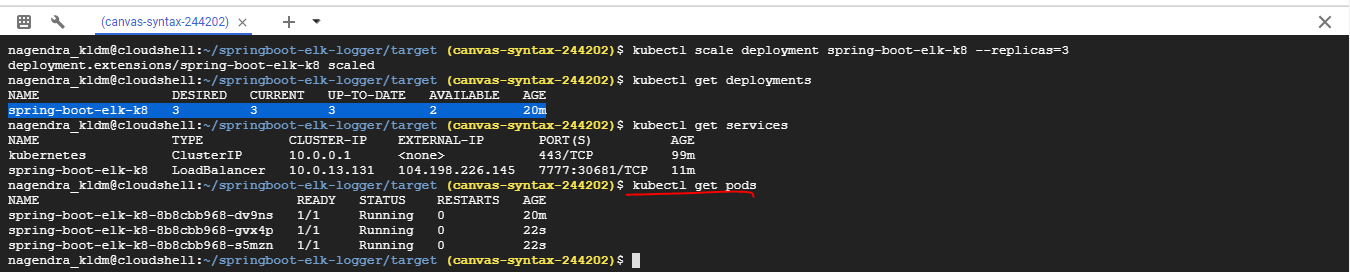
kubectl scale deployment spring-boot-elk-k8 --replicas=3

before command:

****

After command

kubectl scale deployment spring-boot-elk-k8 --replicas=3

****

**Step 11: roolback and undo**

kubectl rollout undo spring-boot-elk-k8 --replicas=3

**Remove Service**

kubectl delete deployment spring-boot-elk

kubectl delete service spring-boot-elk-service

kubectl logs spring-boot-elk-k8-67ff8fd86c-zhn2l

**kubernetes Strategy Types**

* Recreate
* RollingUpdate
* Blue/Green
* Canary

[**recreate**](https://container-solutions.com/kubernetes-deployment-strategies/#kubernetes-recreate)**:** terminate the old version and release the new one

[**ramped**](https://container-solutions.com/kubernetes-deployment-strategies/#kubernetes-ramped)**:** release a new version on a rolling update fashion, one after the other

[**blue/green**](https://container-solutions.com/kubernetes-deployment-strategies/#kubernetes-blue-green)**:** release a new version alongside the old version then switch traffic

[**canary**](https://container-solutions.com/kubernetes-deployment-strategies/#kubernetes-canary)**:** release a new version to a subset of users, then proceed to a full rollout

[**https://github.com/TechPrimers/k8s-spring-boot-example**](https://github.com/TechPrimers/k8s-spring-boot-example)

**https://static.brandonpotter.com/kubernetes/DeploymentBuilder.html**

